RFC: 816

FAULT ISOLATION AND RECOVERY

David D. Clark
MIT Laboratory for Computer Science
Computer Systems and Communications Group
July, 1982

1. Introduction

Occasionally, a network or a gateway will go down, and the sequence of hops which the packet takes from source to destination must change. Fault isolation is that action which hosts and gateways collectively take to determine that something is wrong; fault recovery is the identification and selection of an alternative route which will serve to reconnect the source to the destination. In fact, the gateways perform most of the functions of fault isolation and recovery. There are, however, a few actions which hosts must take if they wish to provide a reasonable level of service. This document describes the portion of fault isolation and recovery which is the responsibility of the host.

2. What Gateways Do

Gateways collectively implement an algorithm which identifies the best route between all pairs of networks. They do this by exchanging packets which contain each gateway's latest opinion about the operational status of its neighbor networks and gateways. Assuming that this algorithm is operating properly, one can expect the gateways to go through a period of confusion immediately after some network or gateway

has failed, but one can assume that once a period of negotiation has passed, the gateways are equipped with a consistent and correct model of the connectivity of the internet. At present this period of negotiation may actually take several minutes, and many TCP implementations time out within that period, but it is a design goal of the eventual algorithm that the gateway should be able to reconstruct the topology quickly enough that a TCP connection should be able to survive a failure of the route.

3. Host Algorithm for Fault Recovery

Since the gateways always attempt to have a consistent and correct model of the internetwork topology, the host strategy for fault recovery is very simple. Whenever the host feels that something is wrong, it asks the gateway for advice, and, assuming the advice is forthcoming, it believes the advice completely. The advice will be wrong only during the transient period of negotiation, which immediately follows an outage, but will otherwise be reliably correct.

In fact, it is never necessary for a host to explicitly ask a gateway for advice, because the gateway will provide it as appropriate. When a host sends a datagram to some distant net, the host should be prepared to receive back either of two advisory messages which the gateway may send. The ICMP "redirect" message indicates that the gateway to which the host sent the datagram is not longer the best gateway to reach the net in question. The gateway will have forwarded the datagram, but the host should revise its routing table to have a different immediate address for this net. The ICMP "destination

unreachable" message indicates that as a result of an outage, it is currently impossible to reach the addressed net or host in any manner. On receipt of this message, a host can either abandon the connection immediately without any further retransmission, or resend slowly to see if the fault is corrected in reasonable time.

If a host could assume that these two ICMP messages would always arrive when something was amiss in the network, then no other action on the part of the host would be required in order maintain its tables in an optimal condition. Unfortunately, there are two circumstances under which the messages will not arrive properly. First, during the transient following a failure, error messages may arrive that do not correctly represent the state of the world. Thus, hosts must take an isolated error message with some scepticism. (This transient period is discussed more fully below.) Second, if the host has been sending datagrams to a particular gateway, and that gateway itself crashes, t en all the other gateways in the internet will reconstruct the topology, but the gateway in question will still be down, and therefore cannot provide any advice back to the host. As long as the host continues to direct datagrams at this dead gateway, the datagrams will simply vanish off the face of the earth, and nothing will come back in return. Hosts must detect this failure.

If some gateway many hops away fails, this is not of concern to the host, for then the discovery of the failure is the responsibility of the immediate neighbor gateways, which will perform this action in a manner invisible to the host. The problem only arises if the very first

gateway, the one to which the host is immediately sending the datagrams, fails. We thus identify one single task which the host must perform as its part of fault isolation in the internet: the host must use some strategy to detect that a gateway to which it is sending datagrams is dead.

Let us assume for the moment that the host implements some algorithm to detect failed gateways; we will return later to discuss what this algorithm might be. First, let us consider what the host should do when it has determined that a gateway is down. In fact, with the exception of one small problem, the action the host should take is extremely simple. The host should select some other gateway, and try sending the datagram to it. Assuming that gateway is up, this will either produce correct results, or some ICMP advice. Since we assume that, ignoring temporary periods immediately following an outage, any gateway is capable of giving correct advice, once the host has received advice from any gateway, that host is in as good a condition as it can hope to be.

There is always the unpleasant possibility that when the host tries a different gateway, that gateway too will be down. Therefore, whatever algorithm the host uses to detect a dead gateway must continuously be applied, as the host tries every gateway in turn that it knows about.

The only difficult part of this algorithm is to specify the means by which the host maintains the table of all of the gateways to which it has immediate access. Currently, the specification of the internet protocol does not architect any message by which a host can ask to be

supplied with such a table. The reason is that different networks may provide very different mechanisms by which this table can be filled in. For example, if the net is a broadcast net, such as an ethernet or a ringnet, every gateway may simply broadcast such a table from time to time, and the host need do nothing but listen to obtain the required information. Alternatively, the network may provide the mechanism of logical addressing, by which a whole set of machines can be provided with a single group address, to which a request can be sent for assistance. Failing those two schemes, the host can build up its table of neighbor gateways by remembering all the gateways from which it has ever received a message. Finally, in certain cases, it may be necessary for this table, or at least the initial entries in the table, to be constructed manually by a manager or operator at the site. In cases where the network in question provides absolutely no support for this kind of host query, at least some manual intervention will be required to get started, so that the host can find out about at least one gateway.

4. Host Algorithms for Fault Isolation

We now return to the question raised above. What strategy should the host use to detect that it is talking to a dead gateway, so that it can know to switch to some other gateway in the list. In fact, there are several algorithms which can be used. All are reasonably simple to implement, but they have very different implications for the overhead on the host, the gateway, and the network. Thus, to a certain extent, the algorithm picked must depend on the details of the network and of the host.

1. NETWORK LEVEL DETECTION

Many networks, particularly the Arpanet, perform precisely the required function internal to the network. If a host sends a datagram to a dead gateway on the Arpanet, the network will return a "host dead" message, which is precisely the information the host needs to know in order to switch to another gateway. Some early implementations of Internet on the Arpanet threw these messages away. That is an exceedingly poor idea.

2. CONTINUOUS POLLING

The ICMP protocol provides an echo mechanism by which a host may solicit a response from a gateway. A host could simply send this message at a reasonable rate, to assure itself continuously that the gateway was still up. This works, but, since the message must be sent fairly often to detect a fault in a reasonable time, it can imply an unbearable overhead on the host itself, the network, and the gateway. This strategy is prohibited except where a specific analysis has indicated that the overhead is tolerable.

3. TRIGGERED POLLING

If the use of polling could be restricted to only those times when something seemed to be wrong, then the overhead would be bearable. Provided that one can get the proper advice from one's higher level protocols, it is possible to implement such a strategy. For example, one could program the TCP level so that whenever it retransmitted a

segment more than once, it sent a hint down to the IP layer which triggered polling. This strategy does not have excessive overhead, but does have the problem that the host may be somewhat slow to respond to an error, since only after polling has started will the host be able to confirm that something has gone wrong, and by then the TCP above may have already timed out.

Both forms of polling suffer from a minor flaw. Hosts as well as gateways respond to ICMP echo messages. Thus, polling cannot be used to detect the error that a foreign address thought to be a gateway is actually a host. Such a confusion can arise if the physical addresses of machines are rearranged.

4. TRIGGERED RESELECTION

There is a strategy which makes use of a hint from a higher level, as did the previous strategy, but which avoids polling altogether. Whenever a higher level complains that the service seems to be defective, the Internet layer can pick the next gateway from the list of available gateways, and switch to it. Assuming that this gateway is up, no real harm can come of this decision, even if it was wrong, for the worst that will happen is a redirect message which instructs the host to return to the gateway originally being used. If, on the other hand, the original gateway was indeed down, then this immediately provides a new route, so the period of time until recovery is shortened. This last strategy seems particularly clever, and is probably the most generally suitable for those cases where the network itself does not provide fault isolation. (Regretably, I have forgotten who suggested this idea to me. It is not my invention.)

5. Higher Level Fault Detection

The previous discussion has concentrated on fault detection and recovery at the IP layer. This section considers what the higher layers such as TCP should do.

TCP has a single fault recovery action; it repeatedly retransmits a segment until either it gets an acknowledgement or its connection timer expires. As discussed above, it may use retransmission as an event to trigger a request for fault recovery to the IP layer. In the other direction, information may flow up from IP, reporting such things as ICMP Destination Unreachable or error messages from the attached network. The only subtle question about TCP and faults is what TCP should do when such an error message arrives or its connection timer expires.

The TCP specification discusses the timer. In the description of the open call, the timeout is described as an optional value that the client of TCP may specify; if any segment remains unacknowledged for this period, TCP should abort the connection. The default for the timeout is 30 seconds. Early TCPs were often implemented with a fixed timeout interval, but this did not work well in practice, as the following discussion may suggest.

Clients of TCP can be divided into two classes: those running on immediate behalf of a human, such as Telnet, and those supporting a program, such as a mail sender. Humans require a sophisticated response to errors. Depending on exactly what went wrong, they may want to

abandon the connection at once, or wait for a long time to see if things get better. Programs do not have this human impatience, but also lack the power to make complex decisions based on details of the exact error condition. For them, a simple timeout is reasonable.

Based on these considerations, at least two modes of operation are needed in TCP. One, for programs, abandons the connection without exception if the TCP timer expires. The other mode, suitable for people, never abandons the connection on its own initiative, but reports to the layer above when the timer expires. Thus, the human user can see error messages coming from all the relevant layers, TCP and ICMP, and can request TCP to abort as appropriate. This second mode requires that TCP be able to send an asynchronous message up to its client to report the timeout, and it requires that error messages arriving at lower layers similarly flow up through TCP.

At levels above TCP, fault detection is also required. Either of the following can happen. First, the foreign client of TCP can fail, even though TCP is still running, so data is still acknowledged and the timer never expires. Alternatively, the communication path can fail, without the TCP timer going off, because the local client has no data to send. Both of these have caused trouble.

Sending mail provides an example of the first case. When sending mail using SMTP, there is an SMTP level acknowledgement that is returned when a piece of mail is successfully delivered. Several early mail receiving programs would crash just at the point where they had received all of the mail text (so TCP did not detect a timeout due to outstanding

unacknowledged data) but before the mail was acknowledged at the SMTP level. This failure would cause early mail senders to wait forever for the SMTP level acknowledgement. The obvious cure was to set a timer at the SMTP level, but the first attempt to do this did not work, for there was no simple way to select the timer interval. If the interval selected was short, it expired in normal operational when sending a large file to a slow host. An interval of many minutes was needed to prevent false timeouts, but that meant that failures were detected only very slowly. The current solution in several mailers is to pick a timeout interval proportional to the size of the message.

Server telnet provides an example of the other kind of failure. It can easily happen that the communications link can fail while there is no traffic flowing, perhaps because the user is thinking. Eventually, the user will attempt to type something, at which time he will discover that the connection is dead and abort it. But the host end of the connection, having nothing to send, will not discover anything wrong, and will remain waiting forever. In some systems there is no way for a user in a different process to destroy or take over such a hanging process, so there is no way to recover.

One solution to this would be to have the host server telnet query the user end now and then, to see if it is still up. (Telnet does not have an explicit query feature, but the host could negotiate some unimportant option, which should produce either agreement or disagreement in return.) The only problem with this is that a reasonable sample interval, if applied to every user on a large system,

can generate an unacceptable amount of traffic and system overhead. A smart server telnet would use this query only when something seems wrong, perhaps when there had been no user activity for some time.

In both these cases, the general conclusion is that client level error detection is needed, and that the details of the mechanism are very dependent on the application. Application programmers must be made aware of the problem of failures, and must understand that error detection at the TCP or lower level cannot solve the whole problem for them.

6. Knowing When to Give Up

It is not obvious, when error messages such as ICMP Destination Unreachable arrive, whether TCP should abandon the connection. The reason that error messages are difficult to interpret is that, as discussed above, after a failure of a gateway or network, there is a transient period during which the gateways may have incorrect information, so that irrelevant or incorrect error messages may sometimes return. An isolated ICMP Destination Unreachable may arrive at a host, for example, if a packet is sent during the period when the gateways are trying to find a new route. To abandon a TCP connection based on such a message arriving would be to ignore the valuable feature of the Internet that for many internal failures it reconstructs its function without any disruption of the end points.

But if failure messages do not imply a failure, what are they for? In fact, error messages serve several important purposes. First, if

they arrive in response to opening a new connection, they probably are caused by opening the connection improperly (e.g., to a non-existent address) rather than by a transient network failure. Second, they provide valuable information, after the TCP timeout has occurred, as to the probable cause of the failure. Finally, certain messages, such as ICMP Parameter Problem, imply a possible implementation problem. In general, error messages give valuable information about what went wrong, but are not to be taken as absolutely reliable. A general alerting mechanism, such as the TCP timeout discussed above, provides a good indication that whatever is wrong is a serious condition, but without the advisory messages to augment the timer, there is no way for the client to know how to respond to the error. The combination of the timer and the advice from the error messages provide a reasonable set of facts for the client layer to have. It is important that error messages from all layers be passed up to the client module in a useful and consistent way.